

# On the Impact of Random Index-Partitioning on Index Compression

M. Feldman  
CS Department, Technion  
Haifa 32000, Israel

moranfe@cs.technion.ac.il

R. Lempel  
Yahoo! Labs  
Haifa 31905, Israel

rlempel@yahoo-inc.com

O. Somekh  
Yahoo! Labs  
Haifa 31905, Israel

orens@yahoo-inc.com

K. Vornovitsky  
CS Department, Technion  
Haifa 32000, Israel  
kolman@cs.technion.ac.il

## ABSTRACT

The performance of processing search queries depends heavily on the stored index size. Accordingly, considerable research efforts have been devoted to the development of efficient compression techniques for inverted indexes. Roughly, index compression relies on two factors: the ordering of the indexed documents, which strives to position similar documents in proximity, and the encoding of the inverted lists that result from the ordered stream of documents. Large commercial search engines index tens of billions of pages of the ever growing Web. The sheer size of their indexes dictates the distribution of documents among thousands of servers in a scheme called local index-partitioning, such that each server indexes only several millions pages. Due to engineering and runtime performance considerations, random distribution of documents to servers is common. However, random index-partitioning among many servers adversely impacts the resulting index sizes, as it decreases the effectiveness of document ordering schemes.

We study the impact of random index-partitioning on document ordering schemes. We show that index-partitioning decreases the aggregated size of the inverted lists logarithmically with the number of servers, when documents within each server are randomly reordered. On the other hand, the aggregated partitioned index size increases logarithmically with the number of servers, when state-of-the-art document ordering schemes, such as lexical URL sorting and clustering with TSP, are applied. Finally, we justify the common practice of randomly distributing documents to servers, as we qualitatively show that despite its ill-effects on the ensuing compression, it decreases key factors in distributed query evaluation time by an order of magnitude as compared with partitioning techniques that compress better.

## Categories and Subject Descriptors

H.3.m [Information Systems]: Information Storage and Retrieval

## General Terms

Experimentation, Performance

## Keywords

Inverted Index, Index Compression, Document Reordering,

Index-Partitioning, Query Processing Time

## 1. INTRODUCTION

The searchable Web spans tens of billions of pages, yet search engine users expect fresh and relevant search results to be delivered within less than a second. Serving simultaneously thousands of queries, web search engines use an *inverted index*, a data structure that supports efficient retrieval of documents containing a set of terms given by the user's query. Due to the huge number of web pages and the resulting amount of data, the index is partitioned over thousands of servers, where each server typically stores and processes the inverted index of only several millions documents [2, 5, 32, 20]. At query time, the query is sent to all servers for processing, and the top results retrieved from all servers are merged to produce the final results, which are returned to the user.

The inverted index data structure contains a *postings list* for each unique term appearing in the corpus. The postings list of term  $t$  consists of the list of document identifiers<sup>1</sup> (docIds) containing  $t$ . The documents within each list are typically sorted by increasing docIds values, and the list is represented by encoding the gaps (called *dGaps*) between successive docIds. Another data structure in an inverted index is the *lexicon*, or *dictionary*, which is a lookup table that for each term  $t$  in the corpus, points to the postings list corresponding to  $t$  [2, 32, 4].

Index size has an important effect on system performance. In addition to the direct reduction in memory and disk space, more compact indexes lead to savings in I/O transfers and increase the hit rate of memory caches, offering an improvement in overall query processing throughput [31, 30]. Consequently, a large body of work has focused on index compaction and compression methods. The structure described above leaves two main degrees of freedom for compression optimization: (a) the assignment of docIds to documents (also referred to as document reordering); and (b) the actual encoding of the dGaps into bits (also referred to as dGap compression [28, 21, 9, 14, 1, 31]). This work focuses

<sup>1</sup>Although terms frequencies and offsets within the document occupy a major portion of modern inverted indexes, we focus here on the documents identifiers only.

on the former.

The basic idea behind an effective docId assignment is to place “similar” documents close to each other, hence, potentially reducing the dGaps since similar documents contain many common terms. Such effective assignment produces highly clustered posting lists where long “runs” of small dGaps are separated by large dGaps. In contrast, a random assignment of docIds would result in dGaps that approximately follow a Geometric distribution within each postings list [12]. The problem of finding the optimal docId assignment can be explicitly expressed in closed form, but is, unfortunately, NP-hard [6]. Therefore, most works on document assignment proposed various heuristics that includes approximations to the traveling salesman problem (TSP), solutions based on clustering algorithms, and solutions based on the natural URL lexicographical ordering of web pages [8, 25, 27, 7, 26, 30, 13].

All aforementioned efforts focused on compacting the inverted index of a single server. However, large corpora are indexed over thousands of servers, with each server handling only several million documents [2, 5, 32, 20]. In order to better balance the number of result documents resulting on each server, thereby decreasing query processing time (see our experiments in Section 6), documents are often distributed randomly among the servers [2, 5, 19, 20]. As first noted in [30], this index-partitioning operation may have a profound effect on document assignment algorithms, since similar documents (e.g., pages of the same web host) are often routed to different servers. This work examines the impact of random index-partitioning on the effectiveness of docId assignment algorithms that aim to compress the inverted index. Our experiments are performed on the 25 million web page TREC .gov2 collection. Our main contributions are the following:

- We showcase the interplay between *random* index-partitioning and compression.
- We quantitatively and analytically show that the performance gap between effective docId assignment heuristics and ineffective ones diminishes as the index is *randomly* partitioned over more servers. For example, with dGap Delta encoding, the total length of the inverted lists actually decreases logarithmically with the number of partitions when docIds are assigned randomly. On the other hand, partitioning causes that size to increase logarithmically with the number of partitions when effective docId assignments such as URL sorting, and clustering with TSP, are applied. Similar trends are reported for dGap block PForDelta encoding as well.
- We study experimentally the factors that make the URL-based assignment perform well in practice. We show that inter-host ordering hardly matters, and that clustering pages by hosts with arbitrary intra-host ordering already brings significant compression benefits.
- We justify the common practice of randomly distributing documents to servers, as we qualitatively show that despite its ill-effects on the ensuing compression, it decreases key factors in distributed query evaluation time by an order of magnitude as compared with the better compressing URL-based partitioning.

The rest of this work is organized as follows. Section 2 provides background and surveys related work. The experimental setup is described in section 3. Experimental results and analytical insight of the impact of partitioning on index sizes are reported in Sections 4 and 5, respectively. The impact of index partitioning on query processing time is considered in Section 6. Finally, we conclude in Section 7.

## 2. BACKGROUND AND PRIOR WORK

### 2.1 Index Partitioning

The sheer size of the Web, the enormous number of search queries, and the required low latency, enforce a distributed inverted index architecture [2, 32, 5]. To support these requirements, both *distribution* and *replication* principles are applied. Replication (or *mirroring*) means making enough identical copies of the system so that the required query load can be served, and is beyond the scope of this work. Distribution means the way the inverted index is partitioned across a collection of nodes.

The two main strategies of partitioning an inverted index are *local index-partitioning* and *global index-partitioning* [3, 22]. According to the local index-partitioning strategy (or *document based partition*), each node is responsible for a disjoint subset of documents in the collection. Each search query is sent to all nodes, each of which returns its top ranking documents for the query. Those lists are then combined in some way to provide the end result. In the global index-partitioning strategy (or *term based partition*), terms are divided into disjoint subsets, such that each node stores postings lists only for a subset of terms.

Due to various theoretical and practical considerations, large-scale search engines follow the local inverted index-partitioning strategy distributing documents across the nodes [5, 3, 22]. Documents can be distributed to nodes using different policies. For example, the *hash distribution policy* allocates documents to nodes in a random fashion by hashing the documents’ URLs to yield a node identifier [19, 20]. Other policies such as *round-robin distribution* are also possible [15].

While random distribution of documents to nodes is used by commercial search engines [2, 5, 19, 20], other distribution schemes were considered in distributed information retrieval systems and peer-to-peer networks. For instance, in [29] (see also [17] for a more recent work) the authors used a two-pass K-means clustering algorithm and a KL-divergence distance metric to organize a document collection into 100 topical clusters (or *shards*) and demonstrated the benefits of selectively searching only a few shards per query. Query logs were used by the authors of [23] (see also [24] for a more recent work) to organize a document collection into multiple shards. Selectively searching shards defined by these clusters was found to be more effective than selectively searching randomly defined shards. Non-random distribution of documents in a distributed search engine was recently considered in [18], where the authors treat the routing of documents to nodes as an online problem in an incremental indexing setting. Under a model where routed documents are appended to the existing index partitions, the authors demonstrate a tradeoff between the compression of a locally-partitioned index and the balanced distribution of documents from the same host across the index partitions.

## 2.2 Inverted Index Compression

As mentioned in the Section 1, we consider a simplified model of an inverted index in which the postings list of term  $t$  holds the docIds containing  $t$ , sorted by increasing value. Denote the list by  $d_1^t, d_2^t, \dots, d_{n_t}^t$ , where  $d_i^t$  denotes the docId of the  $i$ 'th document containing  $t$  out of  $n_t$  such documents. The list is actually represented by encoding the first docId and the sequence of gaps (dGaps) between successive identifiers thereafter, i.e.  $d_1^t, d_2^t - d_1^t, \dots, d_{n_t}^t - d_{n_t-1}^t$ . The two degrees of freedom available for compressing the size of the lists are (a) docId assignment; and (b) dGap encoding. As we focus on the former, we start by briefly reviewing the latter. dGap encoding techniques aim to compress a sequence of integers. The literature contains schemes that encode each gap individually, e.g. Gamma, Delta, Golomb-Rice [28] and Zeta [9] encodings, as well as schemes that encode certain blocks of gaps, e.g. PForDelta [31, 14] and Simple9 [1]. Additionally, the Interpolative Encoding scheme [21] is applied directly on the docIds rather than their dGaps, and works well for clustered term occurrences.

In general, the docId assignment problem seeks a permutation of the documents that minimizes the inverted-index size under a specific dGap encoding scheme. As shown in [6], this problem is NP-hard and various heuristics are used to provide approximations.

The size of an inverted-index is a function of the dGaps, which themselves depend on the way docIds are assigned to documents. All effective dGap encoding techniques represent smaller numbers with fewer bits (about logarithmic in the number value). Hence, assigning docIds in a way which results in smaller dGaps is the key for better compression. This principle drives most works dealing with docId assignment, which accordingly strive to assign close docIds to "similar" documents, i.e. documents that share many terms.

Technically, most works define a graph  $G = (D, E)$ , where  $D$  is the set of documents, and  $E$  is a set of edges representing the similarity between two documents  $d_i, d_j \in D$ . One line of work started by [25] traverses the graph  $G$  to find the maximal weight path connecting all the nodes, assigning docIds accordingly. This is equivalent to the NP-Hard *traveling salesman problem* (TSP). Several TSP approximations were applied for docId assignment in [25, 7, 13]. In [25], a simple greedy nearest neighbors (GNN) approach is used to add one edge at a time. To reduce the computational load, [7] uses singular value decomposition (SVD) to reduce the dimensionality of the term-document matrix. To scale up TSP-based schemes [13] proposes a new framework based on computing TSP on a reduced sparse graph obtained through *locality sensitive hashing*.

In yet another line of work, the nodes of  $G$  are clustered according to their similarity and close docIds are assigned to the nodes (documents) within each cluster. A top-down approach is used in [8], where the whole collection is recursively split into sub-collections, inserting "similar" nodes into the same sub-collections. Then, the sub-collections are merged into an ordered group of nodes. A bottom-up approach called k-scan was proposed in [27]. A hybrid method which combines k-scan clustering and TSP for intra-cluster docId assignment is proposed by [6], and will be used in the experiments reported in this paper.

A different approach, which is both highly scalable and highly effective, was proposed for Web collections in [26].

It assigns docIds according to the lexicographically sorted order of the documents' URLs, utilizing the fact that URL similarity is a strong indicator of document similarity. The scheme was found to perform remarkably well on various Web collections indexed as a whole. It was not, to the best of our knowledge, examined for partitioned collections.

In all the aforementioned works, a heuristic of docId assignment or an encoding of dGaps were empirically tested against several collections and compared to the results of other works. In contrast, [12] analyzes the compressibility of a collection whose documents are generated by a simple probabilistic model in which terms are chosen independently from a given distribution.

## 3. EXPERIMENTAL SETUP

We use the TREC .gov2 Web corpus, a collection of about 25.2 million pages crawled from the gov domain, for the experiments. After parsing, tokenizing (with standard English *stopword removal* and no *stemming*) and removing all empty documents, we are left with 24.9 million documents, 74.5 million distinct terms, and 5,705.2 million postings (distinct term appearances in documents). Whenever we partition the corpus over  $m$  servers, documents are assigned to servers independently and uniformly at random. We then apply some docId assignment and dGap encoding schemes across all servers. Index sizes are reported using the *bits per posting* metric, defined below.

### 3.1 The Bits per Posting Metric

Let a corpus with  $\mathcal{N}$  overall postings be indexed across  $m$  partitions, and let  $T_i$  denote the set of distinct terms on the  $i$ 'th partition. Let  $t$  be a term appearing in  $n_t$  documents in some partition, and denote those docIds by  $1 \leq d_1^t < d_2^t < \dots < d_{n_t}^t$ . Then, the overall size of all postings lists on the  $i$ 'th partition,  $\mathcal{P}_i$ , is given by

$$\mathcal{P}_i = \sum_{t \in T_i} S(d_1^t, d_2^t - d_1^t, \dots, d_{n_t}^t - d_{n_t-1}^t),$$

where  $S(\cdot)$  is the length (in bits) of encoding the given integer sequence. The overall size of the postings across the  $m$  partitions,  $\mathcal{P}$ , is defined as

$$\mathcal{P} = \sum_{i=1}^m \mathcal{P}_i.$$

We experiment with Delta and PForDelta encoding schemes. For Delta encoding

$$S(d_1^t, d_2^t - d_1^t, \dots, d_{n_t}^t - d_{n_t-1}^t) = \delta(d_1^t) + \sum_{j=2}^{n_t} \delta(d_j^t - d_{j-1}^t),$$

where  $\delta(k)$  is the length (in bits) of the Delta encoding of the positive integer  $k$ :

$$\delta(k) = 1 + \lfloor \log_2 k \rfloor + 2 \lfloor \log_2 (1 + \lfloor \log_2 k \rfloor) \rfloor.$$

For PForDelta encoding scheme, each posting list is processed according to the scheme presented in [31], with block length of 128 dGaps, and threshold of 90%. Shorter blocks at the end of long posting lists and short posting lists, down to 64 dGaps are encoded in a similar fashion, while blocks of less than 64 dGaps are simply Delta encoded.

We further define the overhead  $\mathcal{OH}$  of a partitioned index as the space taken by the  $m$  dictionaries of the individual

partitions. Each entry of the  $i$ 'th dictionary is a pointer into the sequence of postings lists on the  $i$ 'th server, and hence requires  $\log_2 \mathcal{P}_i$  bits<sup>2</sup>. Overall,

$$\mathcal{OH} = \sum_{i=1}^m |T_i| \log_2 \mathcal{P}_i.$$

Finally, the *bits per posting* metric comes in two flavors, with and without overhead. Those are simply  $\frac{\mathcal{P} + \mathcal{OH}}{N}$  and  $\frac{\mathcal{P}}{N}$ , respectively.

### 3.2 DocId Assignment Schemes

As stated earlier, we are mainly interested in two aspects: (a) studying the impact of random index-partitioning on the bits per posting metric, and (b) gaining further insight into the power of the URL-based docId assignment scheme. We thus experiment with the following five docId assignment schemes:

**Random assignment (RND):** this method serves as a baseline for comparison purposes.

**URL-based sorting (URL):** following [26], the documents are sorted lexicographically based on their URL<sup>3</sup> and docIds are assigned accordingly.

**Clustering assignment (KSCAN-TSP):** we adopt a procedure presented in [7] where each server's collection is partitioned into  $K$  clusters, and GNN approximation of TSP is used to assign the docIds within each cluster. We set the cluster size (and the number of clusters) to the square root of the server's corpus size, which is known to provide fair results. This heuristic represents, in this work, the state-of-the-art of schemes that are URL-agnostic.

**Intra-host URL-based sorting (IH-URL):** here, the hosts are randomly ordered, and URL-based ordering is kept within the hosts only. This scheme, when compared to the conventional URL scheme, should reveal the contribution of the inter-host lexicographical ordering to the power of URL-based docId assignment.

**Intra-host random assignment (IH-RND):** here, documents of the same host are assigned with consecutive docIds, but both the hosts and the documents within each host are randomly ordered. This scheme should reveal whether the power of URL-based assignment stems merely from the fact that documents of the same host are clustered together, or actually depends on the lexicographic ordering within each host.

When comparing URL-agnostic docId assignment schemes (represented here by KSCAN-TSP) to the URL sorting scheme over partitioned indexes, one hypothesis comes to mind: URL-agnostic schemes should outperform URL assignment when the corpus is highly partitioned, since they have the degree of freedom to arrange documents by similarity that transcends diluted URL patterns.

<sup>2</sup>For simplicity, we assume that individual posting, as well as inverted lists, can start on arbitrary bit boundaries.

<sup>3</sup>The host name components are first inverted, see [26] for details.

## 4. EXPERIMENTAL RESULTS

The bits per posting measure is plotted as function of the number of nodes with and without overhead in Figures 1.a and 1.b, respectively. The curves are plotted for the URL, IH-URL, IH-RND, and RND docId assignment schemes using the full .gov2 corpus and Delta encoding. Figure 1.a demonstrates that without overhead, the aggregated size decreases with the number of nodes for the RND assignment and increases for the URL based schemes (i.e., URL, IH-URL, IH-RND). In particular, the ratio between the sizes of the RND and the URL assignments decreases from 2.2 when no partitioning is applied, to 1.45 when the corpus is partitioning over  $m = 10^3$  nodes. When the overhead is included, the sizes achieved by all schemes increase with the number of nodes, although the performance of URL based schemes degrades at a faster rate than that of the RND scheme. As can be seen, in the region of interest, the curves are approximately linear in  $\log m$ . Beyond this region, as the number of nodes increases, the URL based curves will coincide with that of the RND, and in the limit where each document is placed on a different node the number of bits per posting of all schemes go to one.

In Figures 2.a and 2.b, the bits per posting measure with and without overhead is plotted as function of the number of nodes, respectively. The curves are plotted for the URL, IH-URL, IH-RND, RND, and KSCAN-TSP docId assignment schemes using 3 million pages taken as a URL-continuous bulk from .gov2 corpus<sup>4</sup> and Delta encoding. It can be seen that the compression achieved by the KSCAN-TSP scheme behaves similarly to that of the URL based schemes, and increases with the number of nodes. We note that although KSCAN-TSP is expected to perform as the RND scheme in the limit, where the number of nodes is large, one could expect that KSCAN-TSP will degrade more gracefully with the number of nodes than URL. This is since KSCAN-TSP (and other state-of-the-art schemes) have an additional degree of freedom over URL sorting, in their ability to reorder the local documents after partitioning. However, as seen here, both URL and KSCAN-TSP degrade at similar rates.

Comparing the figures 1 and 2 produced for the full .gov2 corpus and for the 3 million document sub-corpus respectively, using Delta encoding, reveals that the shapes of the curves and the relations between them are similar. This strengthens our conjecture that the same behavior also hold for web scale collections.

Turning to PForDelta encoding, Figures 3.a and 3.b plot the bits per posting measure as function of the number of nodes with and without overhead, respectively. The curves are plotted for the URL, IH-URL, IH-RND, and RND docId assignment schemes using the full .gov2 corpus and PForDelta encoding. In general, the trends visible for Delta encoding and all docId assignment schemes (Figure 2) are also visible here for the PForDelta encoding curves. Nevertheless, comparing figures 1 and 3 it is observed that while the RND assignment curve decreases in a lower rate than that of the Delta encoding curve, the URL based sorting curves are increasing in a higher rate than those of the Delta encoding<sup>5</sup>.

Another observation, visible in all figures, relates to the

<sup>4</sup>A smaller corpus is used due to run time considerations of the KSCAN-TSP scheme.

<sup>5</sup>Also visible is the superiority of Delta encoding over the specific variant of the PForDelta scheme used here, which is consistent with the results reported in [30].

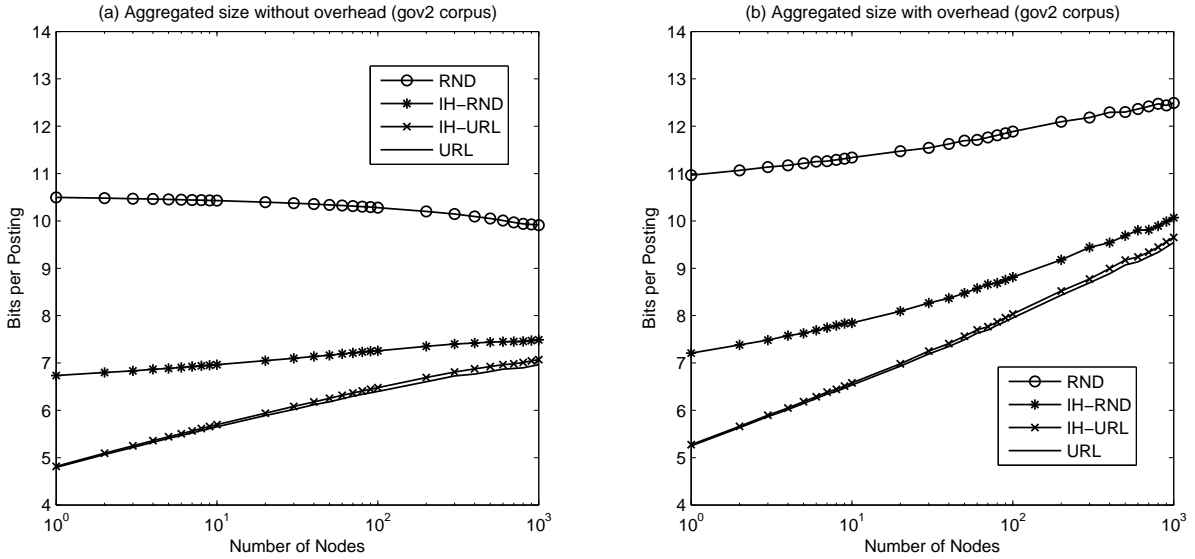


Figure 1: Bits per posting as function of the number of nodes for different docId assignment schemes and Delta encoding applied to .gov2 corpus, (a) without and (b) with overhead.

true nature of URL sorting. By merely clustering each host’s documents together, the IH-RND scheme achieves 75% to 85% (for Delta encoding over the range of node numbers) of the performance improvement of URL sorting over random assignment. Moreover, the performance of IH-URL is almost identical to that of URL. To be precise, URL is slightly better than IH-URL (about 1% on the average) over the range of node numbers. We conclude that the impressive effectiveness of URL sorting for Web corpora such as .gov2, stems mostly from the act of clustering documents of the same host together (i.e., IH-RND scheme). Keeping the lexicographical order within each host is the secondary contributor to the effectiveness of the URL scheme, and when combined with host clustering (i.e., IH-URL), it provides almost identical performance to that of URL sorting. On the other hand, keeping the lexicographical order across hosts has a negligible effect, and hosts can be placed randomly without degrading the URL scheme’s effectiveness. These conclusions, while of little practical implication, provide some insight into the true nature of URL sorting.

We note that these results were all generated under random document distribution to nodes (see Section 2.1). Experimental results (not presented here) with round-robin distribution did not produce qualitatively different results. In addition, experimental results (also not presented here) reveal a small variance between multiple runs. Hence, the corpora used are large enough that self averaging is dominant. Hence, multi runs are redundant and all the presented results are of a single run experiments.

## 5. ANALYTICAL INSIGHT ON RESULTS

This section provides analytical and illustrative explanations to some of our experimental results. In particular, we prove that for random docId assignment and individual dGap logarithmic encoding (e.g., Delta encoding), the average aggregated index size (ignoring overhead) is a non-increasing function of the number of partitions. Conversely, for URL (and IH-URL) assignment and individual dGap

logarithmic encoding, we demonstrate that partitioning increases the aggregated size. We note that the impact of index-partitioning on docId assignment and PForDelta encoding is much harder to explain since this encoding scheme works in blocks of dGaps, and is left for further study.

### Index-Partitioning and Random docId Assignment

Our model for index-partitioning under random docId assignment is as follows. Let there be  $|D|$  documents and  $m$  nodes, and assume for simplicity that  $m$  divides  $|D|$  and that the documents are evenly distributed across the nodes. We first draw uniformly at random (u.a.r.) a permutation  $\pi$  over the documents, and then draw an equal partitioning (denoted by  $g^m$ ) of  $m$  sets of  $\frac{|D|}{m}$  documents each, also u.a.r.

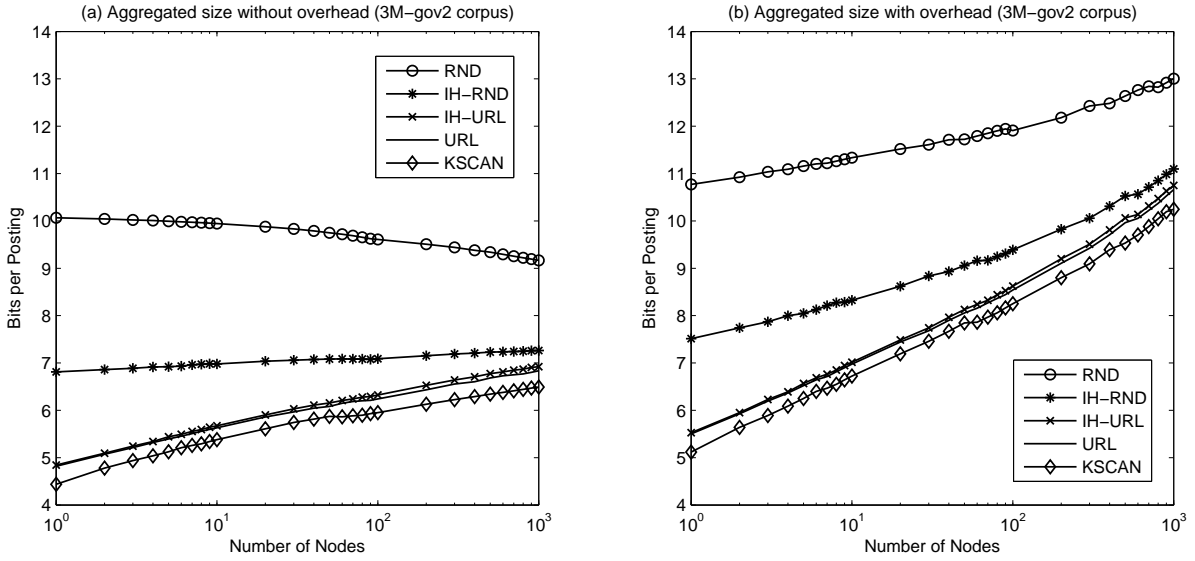
There are  $|D|! (m!)^{-\frac{|D|}{m}}$  such partitions. The document sets get assigned to the servers, with the internal order on each server respecting (being consistent with)  $\pi$ . We aim to prove that the following expectation, denoted  $\Delta^m$ , is non-negative:

$$\Delta^m = E_{\pi, g}[\mathcal{P}(\pi) - \mathcal{P}^m(\pi, g)] \geq 0,$$

where  $\mathcal{P}(\pi)$  denotes the length of all postings lists when the documents are ordered by  $\pi$  and indexed on a single node, and  $\mathcal{P}^m(\pi, g)$  denotes the aggregated length of all postings lists when the documents are partitioned by  $g$  into  $m$  nodes, with the internal order in each node respecting  $\pi$ . Now,

$$\begin{aligned} \Delta^m &= \sum_g \frac{(m!)^{\frac{|D|}{m}}}{|D|!} \sum_{\pi} \frac{1}{|D|!} [\mathcal{P}(\pi) - \mathcal{P}^m(\pi, g)] \\ &= \frac{(m!)^{\frac{|D|}{m}}}{(|D|!)^2} \sum_g \sum_{\pi} [\mathcal{P}(\pi) - \mathcal{P}^m(\pi, g)]. \end{aligned}$$

Looking at the inner sum, observe that for a fixed partition  $g$  and every permutation  $\pi$  there exists a single permutation  $\tilde{\pi}$  that represents the concatenation of the  $m$  partial permutations. Furthermore, for a fixed  $g$ , the mapping between  $\pi$  and  $\tilde{\pi}$  is 1:1 and onto. We now define the  $m$ -slice partitioning of a  $|D|$ -sized permutation, denoted  $g_m$ , as the



**Figure 2: Bits per posting as function of the number of nodes for different docId assignment schemes and Delta encoding applied to a bulk of 3 million URL-continuous documents from .gov2 corpus, (a) without and (b) with overhead.**

process of assigning the first  $\frac{|D|}{m}$  documents to the first node, and so on, until assigning the last  $\frac{|D|}{m}$  documents to the  $m$ 'th node. By definition applying  $g$  on  $\pi$  is equivalent to applying  $g_m$  on  $\tilde{\pi}$ , and so:

$$\begin{aligned} \Delta^m &= E_{\pi, g}[\mathcal{P}(\pi) - \mathcal{P}^m(\pi, g)] \\ &= \frac{(m!)^{\frac{|D|}{m}}}{(|D|!)^2} \sum_g \sum_{\pi} [\mathcal{P}(\pi) - \mathcal{P}^m(\tilde{\pi}, g_m)] . \end{aligned}$$

As  $\pi$  goes over all  $|D|!$  permutations, so does  $\tilde{\pi}$ , and thus

$$\begin{aligned} \Delta^m &= \frac{(m!)^{\frac{|D|}{m}}}{(|D|!)^2} \sum_g \sum_{\pi} [\mathcal{P}(\tilde{\pi}) - \mathcal{P}^m(\tilde{\pi}, g_m)] \\ &= \frac{1}{|D|!} \sum_{\tilde{\pi}} [\mathcal{P}(\tilde{\pi}) - \mathcal{P}^m(\tilde{\pi}, g_m)] . \end{aligned}$$

To conclude the proof, we argue that  $\forall \tilde{\pi}, \mathcal{P}(\tilde{\pi}) - \mathcal{P}^m(\tilde{\pi}, g_m) \geq 0$ . Since the transformation involves only slicing, all intra-slice dGaps remain the same for the original and partitioned indexes (or slices), while dGaps bridging across slices are shorter within the partitioned indexes. In expectation, the bridging dGaps are halved by the slicing process, and assuming a logarithmic encoding function (e.g. Delta encoding), about 1 bit is gained on account of each bridging dGap. As the number of nodes (or slices)  $m$  increases, more dGaps bridge across slices. Hence, the expected difference  $\Delta^m$  does not decrease with  $m$ .

### Index-Partitioning and URL Sorting

Ideally, a postings list following URL-based assignment includes runs of small dGaps separated by long dGaps. To illustrate the impact of index-partitioning into  $m$  nodes on the performance of URL sorting, consider a specific posting list which begins with a single large dGap of  $N_1$ , followed by a run of  $R$  dGaps of 1, another large dGap of  $N_2$ , and another run of  $R$  dGaps of 1, with  $N_1, N_2 \gg R \gg$

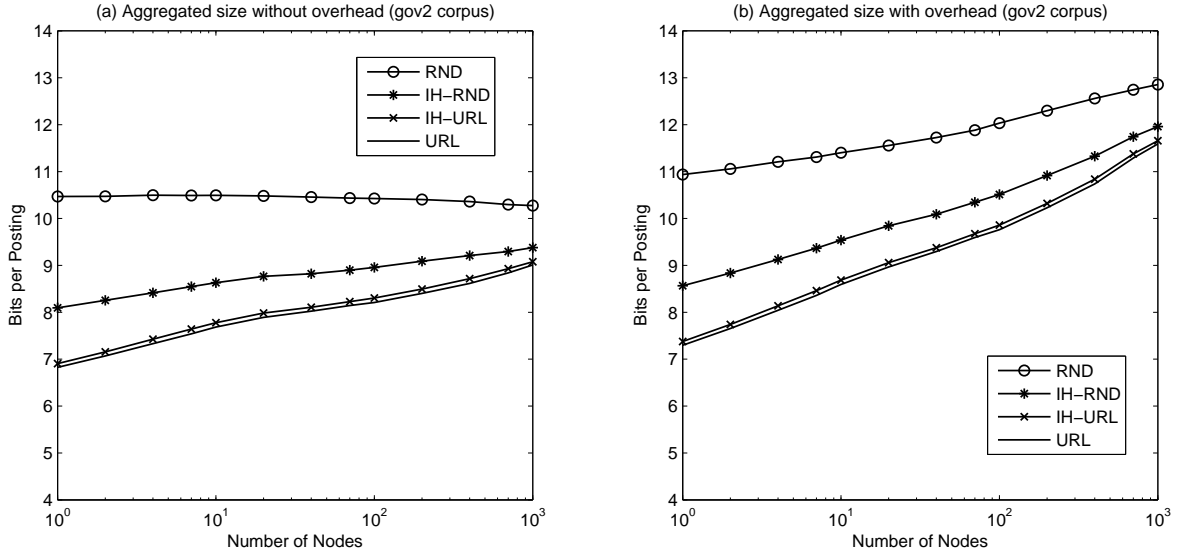
$m$ . Under Delta encoding, the size of the postings list is  $\delta(N_1) + \delta(N_2) + 2R\delta(1)$ . It is easily verified that the average aggregated size after partitioning is approximated by  $m[\delta(N_1/m) + \delta(N_2/m) + 2(R/m)\delta(1)]$ . Hence, the difference in the postings list sizes after and before partitioning into  $m$  nodes is approximately  $m[\delta(N_1/m) + \delta(N_2/m)] - (\delta(N_1) + \delta(N_2))$ . Since Delta encoding behaves logarithmically, partitioning increases the average overall size by approximately  $(m-1)(\log_2 N_1 + \log_2 N_2) - 2m \log_2 m$ . Obviously, this oversimplified example does not represent all cases, but it teaches us that for URL sorting (and IH-URL sorting), the encoding of the partitioned large dGaps of the original list causes its aggregated size to increase.

## 6. DOCUMENT DISTRIBUTION SCHEMES AND QUERY PROCESSING TIME

The previous sections demonstrated the deleterious effect of random distribution of documents to nodes, on the aggregated index sizes. This section examines the impact of document distribution schemes on other factors affecting query processing time, and demonstrates the significant benefits of random distribution – which make it the industry standard [2, 5, 20, 19]. In particular, we qualitatively show that random distribution results in faster query processing than that achieved by the better compressing URL-based distribution scheme.

### 6.1 Surrogates for Query Processing Time

In order for our ensuing experiments and qualitative analysis to be independent of specific retrieval algorithms or computational platforms, we use surrogate measures that are highly correlated with query evaluation time, for both *disjunctive* and *conjunctive* query models. In what follows, let  $q = \{t_1, \dots, t_k\}$  be a  $k$ -term query, and let  $\ell(t)$  denote the number of postings in term  $t$ 's postings list. In disjunctive queries, disregarding various pruning and early termination schemes, retrieval algorithms must scan all lists to fully eval-



**Figure 3: Bits per posting as function of the number of nodes for different docId assignment schemes and PForDelta encoding applied to .gov2 corpus, (a) without and (b) with overhead.**

uate the query. Hence, a surrogate measure for the running time of a disjunctive query on a particular index partition would be  $\sum_{t \in q} \ell(t)$ . In a locally-partitioned index among  $m$  nodes, query evaluation (again, disregarding timeout or pruning policies) must wait for the slowest partition to finish evaluating the query. Hence, we approximate the running time of  $q$  on  $m$  nodes in disjunctive semantics,  $\mathcal{T}_d(q)$ , by

$$\mathcal{T}_d(q) = \max_{j=1, \dots, m} \sum_{t \in q} \ell_j(t),$$

where  $\ell_j(t)$  denotes the length of  $t$ 's postings list on the  $j$ 'th partition. Moving to conjunctive models, queries are typically evaluated by join-flavored algorithms [10, 11] that rely on the ability to skip portions of postings lists where matches are known not to exist [28]<sup>6</sup>. Therefore, our surrogate for  $q$ 's running time on a particular index partition is the length of the postings list of its rarest term,  $\min_{t \in q} \ell(t)$ . In a distributed setting, the slowest partition dictates that

$$\mathcal{T}_c(q) = \max_{j=1, \dots, m} \min_{t \in q} \ell_j(t).$$

We stress that we do not claim that these measures *equal* query running times - only that for most retrieval algorithms on RAM-resident indexes, they represent reasonable surrogates that are correlated with running times.

## 6.2 Experimental Evaluation

We again use the TREC .gov2 corpus (see Section 3), and distribute its documents to servers using random distribution (RND), and two flavors of URL-based distribution. First, vanilla URL distribution (URL), where all documents are ordered lexicographically according to their URL and then evenly sliced and routed to servers; second, IH-URL distribution - where hosts are randomly ordered and same host documents are lexicographically sorted according by URL before being evenly sliced and routed to servers.

<sup>6</sup>We ignore the small overhead that such skipping mechanisms add to the lengths of the postings lists.

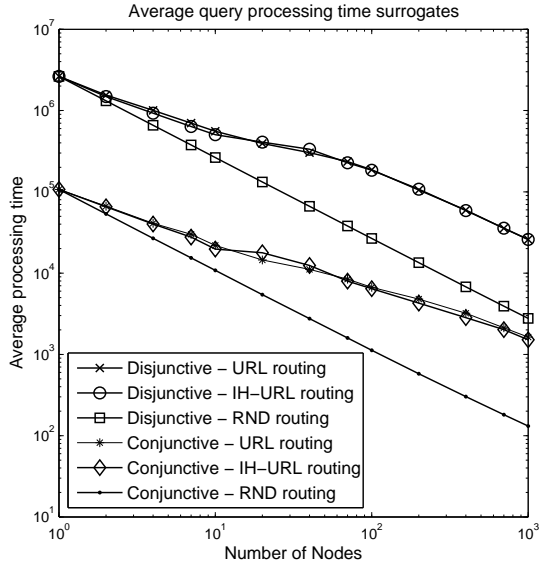
We use the 150 queries of TREC topics 701-850, whose average length is 3.1 terms, and report the average  $\mathcal{T}_c = \frac{1}{150} \sum_{q=701}^{850} \mathcal{T}_c(q)$  and the similarly defined average  $\mathcal{T}_d$  resulting from the three document distribution schemes over all queries, to qualitative compare their average query processing time.

Figure 4 plots the  $\mathcal{T}_c$  and  $\mathcal{T}_d$  curves for the two query types as functions of the number of servers, for the three document distribution schemes RND, URL, and IH-URL. The figure reveals the significant benefit of RND over the URL-based assignment schemes in terms of query processing time, and furthermore that the difference between the curves induced by RND and the URL-based schemes increases with the number of servers. In particular, RND induces  $\mathcal{T}_c$  and  $\mathcal{T}_d$  curves that are an order of a magnitude lower (i.e. faster) than those induced by the URL-based schemes at  $m = 1000$  servers. A closer inspection of the RND curves reveal that their slope is approximately  $-1$  in a  $\log - \log$  scale. Hence, RND scheme induced  $\mathcal{T}_c$  and  $\mathcal{T}_d$  are proportionally inverse to the number of servers:  $\mathcal{T}_c, \mathcal{T}_d \propto \frac{1}{m}$ . Finally, note the similar performance demonstrated by the two URL-based schemes, which is explained by the weak inter-host document similarity already observed in Section 4.

The degradation in query processing time obtained by the URL-based distribution schemes can be intuitively explained by the fact that same host documents are similar (which is good for reducing the index size) and share many terms. Hence, placing them on the same partition yields unbalanced posting lists which increases query processing time due to the maximum operation included in the calculation of both  $\mathcal{T}_c$  and  $\mathcal{T}_d$ .

## 6.3 Analytical Evaluation

This subsection analytically explains why the slopes of RND's  $\mathcal{T}_c$  and  $\mathcal{T}_d$  curves are inversely proportional to the number of servers  $m$ . For simplicity, we assume the document generation model of [12], in which terms are picked to document independently. Hence, for a disjunctive query  $q$ ,



**Figure 4: Average query processing time surrogates vs. number of servers for different document routing schemes.**

we can equate  $\mathcal{T}_d(q)$  to the most occupied among  $m$  urns (servers) when  $b_q = \sum_{t \in q} df(t)$  balls (postings) are randomly tossed to the urns ( $df(t)$  denotes the document frequency of term  $t$  in the entire corpus) [16].

**Proposition 1** For any  $0 < \epsilon < 1$  and  $\delta_\epsilon \leq \delta < 2e - 1$ ,

$$\text{Prob} \left( \mathcal{T}_d(q) \in \left[ \frac{b_q}{m}, \frac{b_q}{m} (1 + \delta) \right] \right) > 1 - \epsilon,$$

$$\text{where } \delta_\epsilon \triangleq \sqrt{\frac{4m}{b_q} \log \frac{m}{\epsilon}}.$$

PROOF. Let  $b_q$  balls be tossed randomly into  $m$  urns, and let  $x_j$  be the number of balls in the  $j$ th urn. In addition, denote the expected number of balls in an urn by  $\mu \triangleq \frac{b_q}{m}$ , and let  $x_{max} = \max_{j=1, \dots, m} x_j$  be the maximal number of balls falling into some urn. Setting  $\alpha \triangleq \mu(1 + \delta)$  for some  $\delta > 0$  we can write

$$\begin{aligned} \text{Pr}(x_{max} \geq \alpha) &= \text{Pr} \left( \bigcup_{j=1}^m x_j \geq \alpha \right) \\ &\leq m \text{Pr}(x_1 \geq \alpha) \leq m e^{-\frac{\mu \delta^2}{4}}, \end{aligned}$$

where the first inequality is due to the union bound, and the second inequality is achieved by applying Chernoff's bound and holds for  $\delta < 2e - 1$ . Forcing the last term of the previous expression to be smaller than  $0 < \epsilon < 1$ , we have that  $\delta$  must also satisfy

$$\delta \geq \sqrt{\frac{4m}{b_q} \log \frac{m}{\epsilon}}.$$

The proof is completed by recalling that  $x_{max} \geq \mu$ .  $\square$

The average number of postings  $b_q$  for the  $N = 150$  TREC topics 701-850 is about  $2.6 \times 10^6$ , whereas the number of

servers in this experiment does not exceed  $10^3$ . Therefore, we can apply Prop. 1 and write

$$\mathcal{T}_d = \frac{1}{N} \sum_q \mathcal{T}_d(q) \approx \frac{1}{m N} \sum_q b_q = \frac{1}{m} \left( \frac{1}{N} \sum_q \sum_{t \in q} df(t) \right).$$

Hence,  $\mathcal{T}_d$  is inversely proportional to the number of servers  $m$ , as observed in Fig. 4.

The expression  $\mathcal{T}_c$  corresponding to conjunctive queries involves a max-min operation, which complicates the exact analysis. Therefore, we analyze an upper bound which is obtained by only considering the rarest term of each query. In this case,  $\mathcal{T}_c(q)$  equals the maximum urn occupancy of a simple urn model where  $b_q = \min_{t \in q} df(t)$  balls are randomly tossed into  $m$  urns. Since the average of  $b_q$  for the  $N = 150$  queries of TREC topics 701-850 is about  $10^5$  – still at least two orders of magnitude over the number of servers  $m$ , we can apply Prop. 1 and write

$$\mathcal{T}_c = \frac{1}{N} \sum_q \mathcal{T}_c(q) \approx \frac{1}{m N} \sum_q b_q = \frac{1}{m} \left( \frac{1}{N} \sum_q \min_{t \in q} df(t) \right).$$

Hence, as in the disjunctive case,  $\mathcal{T}_c$  is inversely proportional to the number of servers  $m$ , as observed in Fig. 4. It is noted that this approximated upper bound is tight since it has  $-1$  slope in  $\log - \log$  scale, and it includes the same constant as the experimented curve for  $m = 1$ .

## 7. CONCLUSIONS

We studied the impact of random index-partitioning on the performance of various docId assignment techniques, and demonstrated the deleterious effect of random index-partitioning in terms of the aggregated size of the partitioned index. We conjecture that our findings, based on the TREC.gov2 corpus and backed by some analysis, also hold at web scale – that randomized index-partitioning generates local collections that state-of-the-art ordering schemes can compress with relatively minor improvement over random ordering. The main reason for that, is that random index-partitioning causes pages of the same web host to be scattered over many nodes, resulting in local collections that are “sparse” in terms of URL continuity and that include few documents having high similarity with each other. Therefore, it follows that from a pure index size perspective, global index-partitioning where terms (instead of documents) are partitioned between nodes will compress better than the industry standard of randomized local index-partitioning. We also show via experimental evaluation that most of the effectiveness of URL sorting is achieved by merely clustering same host documents together. Moreover, we demonstrate that while URL sorting the documents within the hosts does yield additional improvement, keeping the lexical URL ordering of the hosts brings only negligible benefit. Lastly, we demonstrate the benefits of the industry standard random partitioning of documents to servers in terms of query processing time, over URL-based partitioning schemes.

## 8. REFERENCES

- [1] V. Anh and A. Moffat. Inverted index compression using word-aligned binary codes. *Journal of Information Retrieval (IR)*, 8(1):151–166, Jan. 2005.
- [2] A. Arasu, J. Cho, H. Garcia-Molina, A. Paepcke, and S. Raghavan. Searching the web. *ACM Trans. on Internet Technology (TOIT)*, 1(1):2–43, Aug. 2001.



- [3] C. Badue, R. Baeza-Yates, B. Ribeiro-neto, and N. Ziviani. Distributed query processing using partitioned inverted files. In *Proc. of the 9th String Processing and Information Retrieval Symposium (SPIRE'01)*, pages 10–20, Laguna de San Rafael, Chile, Nov. 12–16 2001.
- [4] R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval: The Concepts and Technology behind Search (2nd Edition)*. ACM Press Books/Addison Wesley Professional, 2011.
- [5] L. A. Barroso, J. Dean, and U. Hözlze. Web search for a planet: The Google cluster architecture. *IEEE Micro*, 23(2):22–28, Apr. 2003.
- [6] R. Blanco. *Index Compression for Information Retrieval Systems*. PhD thesis, University of A Coruña, 2008.
- [7] R. Blanco and R. Barreiro. TSP and cluster-based solutions to the reassignment of document identifiers. *Journal of Information Retrieval (IR)*, 9(4):499–517, Sep. 2006.
- [8] D. Blandford and G. Blelloch. Index compression through document reordering. In *Proc. Data Compression Conference (DCC'02)*, Snowbird, UT, 2002.
- [9] P. Boldi and S. Vigna. Codes for the world wide web. *Internet Mathematics*, 2(4):405–427, 2005.
- [10] A. Broder, D. Carmel, M. Herscovichi, A. Soffer, and J. Zien. Efficient query evaluation using a two-level retrieval process. In *Twelfth International Conference on Information and Knowledge Management (CIKM 2003)*, New Orleans, LA, USA, pages 426–434, November 2003.
- [11] A. Z. Broder, N. Eiron, M. Fontoura, M. Herscovici, R. Lempel, J. McPherson, R. Qi, and E. J. Shekita. Indexing of shared content in information retrieval systems. In *Proc. 10th International Conference on Extending Database Technology (EDBT 2006)*, pages 313–330, March 2006.
- [12] F. Chierichetti, R. Kumar, and P. Raghavan. Compressed web indexes. In *Proc. of the 18th International World Wide Web Conference (WWW'09)*, pages 451–460, Madrid, Spain, Apr. 20–24 2009.
- [13] S. Ding, J. Attenberg, and T. Suel. Scalable techniques for document identifier assignment. In *Proc. 19th International World Wide Web Conference (WWW'10)*, pages 311–320, Raleigh, NC, Apr. 26–30 2010.
- [14] S. Hámán. Super-scalar database compression between ram and cpu-cache. Master's thesis, Centrum voor Wiskunde en Informatica (CWI), 2005.
- [15] J. Hirai, S. Raghavan, H. Garcia-Molina, and A. Paepcke. WebBase: A repository of web pages. In *Proc. of the 9th International World Wide Web Conference (WWW'00)*, pages 277–293, Amsterdam, Netherlands, May 15–19 2000.
- [16] N. L. Johnson and S. I. Kotz. *Urn Models and their Application*. John Wiley & Sons, Inc., 1977.
- [17] A. Kulkarni and J. Callan. Topic-based index partitions for efficient and effective selective search. In *Proc. of the 8th International Workshop on Large-Scale Distributed Systems for Information Retrieval (LSDS-IR'10)*, Geneva, Switzerland, Jul. 23 2010.
- [18] G. Lavee, E. Liberty, R. Lempel, and O. Somekh. Inverted index compression via online document routing. *Proc. 20th International World Wide Web Conference (WWW'2011)*, to appear, March 2011.
- [19] R. Lempel and S. Moran. Optimizing result prefetching in web search engines with segmented indices. *ACM Transaction on Internet Technology (TOIT)*, 4(1):31–59, February 2004.
- [20] C. D. Manning, P. Raghavan, and H. Schtze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.
- [21] A. Moffat and L. Stuiver. Binary interpolative coding for effective index compression. *Journal of Information Retrieval (IR)*, 3(1):25–47, Jul. 2000.
- [22] A. Moffat, W. Webber, J. Zobel, and R. Baeza-Yates. A pipelined architecture for distributed text query evaluation. *Inf. Retr.*, 10(3):205–231, 2007.
- [23] D. Puppín, F. Silvestri, and D. Laforenza. Query-driven document partitioning and collection selection. In *Proc. of the 1st international conference on Scalable information systems (InfoScale'06)*, Hong Kong, May 30–Jun. 1 2006.
- [24] D. Puppín, F. Silvestri, R. Perego, and R. Baeza-Yates. Tuning the capacity of search engines: Load-driven routing and incremental caching to reduce and balance the load. *ACM Transactions on Information Systems (TOIS)*, 28(2), May 2010.
- [25] W. Shieh, T. Chen, J. Shann, and C. Chung. Inverted file compression through document identifier reassignment. *Journal of Information Processing and Management*, 39(1):117–131, 2003.
- [26] F. Silvestri. Sorting out the document identifier assignment problem. In *Proc. of 29th European Conference on Information Retrieval (ECIR'07)*, pages 101–112, Rome, Italy, Apr. 2–5 2007.
- [27] F. Silvestri, R. Perego, and S. Orlando. Assigning document identifiers to enhance compressibility of web search engines indexes. In *Proc. of the 2004 ACM Symposium on Applied Computing (SAC'04)*, pages 600–605, Nicosia, Cyprus, Mar. 14–17 2004.
- [28] I. Witten, A. Moffat, and T. Bell. *Managing Gigabytes*. Morgan Kaufmann, 1999.
- [29] J. Xu and W. B. Croft. Cluster-based language models for distributed retrieval. In *Proc. of the 22nd conference on Research and development in Information Retrieval (SIGIR'99)*, Berkeley, CA, Aug. 15–19 1999.
- [30] H. Yan, S. Ding, and T. Suel. Inverted index compression and query processing with optimized document ordering. In *Proc. 18th International World Wide Web Conference (WWW'09)*, Madrid, Spain, Apr. 20–24 2009.
- [31] J. Zhang, X. Long, and T. Suel. Performance of compressed inverted list caching in search engines. In *Proc. of the 17th International World Wide Web Conference (WWW'08)*, pages 387–396, Beijing, China, Apr. 21–25 2008.
- [32] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Computing Surveys*, 38(2), Jul. 2006.